SAND2013-2712P

Execution Models for Exascale – A Bottom-Up Approach – (EMBU)

Sandia National Laboratories
Robert Clay
Gilbert Hendry

Indiana University
Thomas Sterling
Matt Anderson

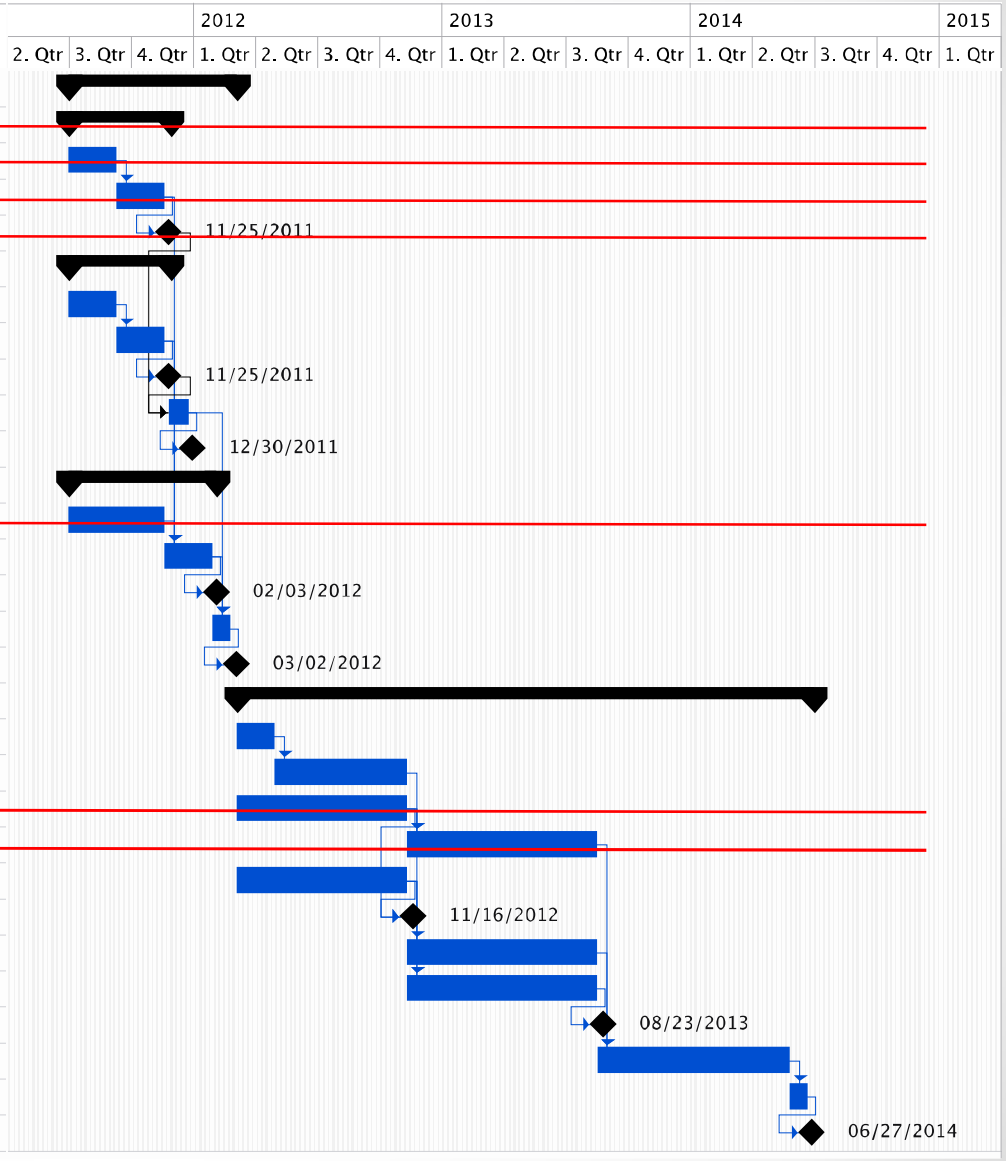
Sandia National Laboratories

Zero'th Order Changes in Plan

- Curt Janssen, PI, left to go work for Google shortly after project started. Robert Clay became EMBU PI, while John Shalf acted as overall coordinator across the three projects.
- LBL's EMBU budget was zero'd out (by HQ) early on. Loss of key staff at LBL, and no funding, led to LBL effectively dropping out as active partner and collaborator.
- Leaves EMBU with missing important piece of co-design process involving low-level simulation and emulation

Project Deliv's (original SOW)

	Task Name	WBS-Code
1	**Phase 1**	**1**
2	~~**Node-level execution model simulation**~~	~~**1.1**~~
3	~~Implement node-level EXEMT~~	~~1.1.1~~
4	~~Implement miniapp node codes~~	~~1.1.2~~
5	~~Demonstrate node-level miniapp simulations~~	~~1.1.3~~
6	**Network-level execution model simulation**	**1.2**
7	Implement network-level EXEMT	1.2.1
8	Implement miniapp skeleton codes	1.2.2
9	Demonstrate network-level miniapp simulations	1.2.3
10	Write preliminary report	1.3
11	Deliver preliminary report	1.4
12	**Execution model co-design**	**1.5**
13	~~Develop processor features for EXEMT~~	~~1.5.1~~
14	Perform simulation studies and parameterize models	1.5.2
15	Demonstrate execution model co-design using miniapps	1.5.3
16	Write report	1.6
17	Deliver phase 1 report and software	1.7
18	**Phase 2**	**2**
19	Identify additional execution models	2.1
20	Implement necessary support in EXEMT	2.2
21	~~Implement miniapp node codes~~	~~2.3~~
22	~~Implement miniapps node codes with alternate exec models~~	~~2.4~~
23	Implement miniapp skeletons	2.5
24	Deliver miniapps and skeletons	2.6
25	Implement miniapp skeletons with alternate exec models	2.7
26	Full system simulation studies and process refinement	2.8
27	Demonstrate full system simulation with phase 2 miniapps	2.9
28	Full system simulation studies with alternate exec models	2.10
29	Write report	2.11
30	Deliver phase 2 report and software	2.12



Sandia National Laboratories

Budget

Planned

	FY11	FY12	FY13	FY14	Total
SNL	335,000	112,000	335,000	218,000	$1,000,000
LBL	55,000	165,000	165,000	115,000	$500,000
IU		150,000	131,900		$281,900
Total					

Spent

	FY11	FY12	FY13	FY14	Total
SNL	12,853	134,140	373,861	479,146	$1,000,000
LBL	55,000	165,000	165,000	115,000	$500,000
IU		68,104	89,139	124,657	$281,900
Total					

Project Management

- Subcontracting IU for HPX work
- At beginning, was having telecons with entire group
- Now mostly interacting with ISI/LBL

- Future management plan: more of the same as of late

EMBU Refocus

Our aim is to perform focused experiments around features of potential future execution models that will steer research. Specifically, we are going to use coarse-grain simulation to study the characteristics of execution models with respect to architectural and runtime features, with a focus on how applications scale in parallel execution to large node counts.

1. What are the characteristics of applications using implementations of different execution models as they scale?
2. What system-level architectural features are critical, given implementations of different execution models
3. What software features are critical in implementations of different execution models related to parallel scalability

Simulation approach (in general)

- Simulation can be used for:
  - rapid prototyping of hardware designs, software APIs and implementations, and runtime features
  - introspection into system features that it is difficult or impossible to do otherwise
  - developing models of hardware/software features that don't exist yet
  - running on scales that don't exist yet

Our Simulator-based Approach

- SST/macro is a coarse-grain simulator driven by online code execution with the goal of faster system-level design-space exploration
  - Tools and processes to support this, like formal UQ methods/people
- Implement rough versions of different execution models in SST/macro
  - capture what we want to observe, nothing more
  - is easier, faster than doing the real thing
  - allows us to easily scale, and introspect
  - SST/macro and HPX implementation can be found at:
    - bitbucket.org/ghendry/sstmacro
- Port applications to SST/macro execution model implementations
  - APIs are the same (between SST/mac and real thing)
  - skeletonize, for faster execution and rapid prototyping
  - Currently have GTC, HPCCG

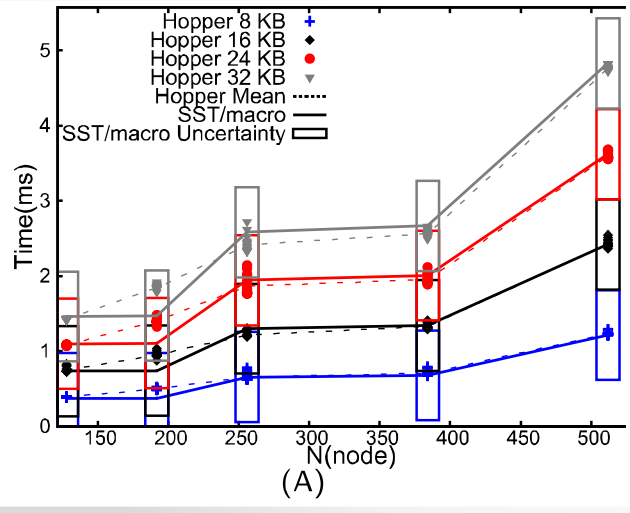
About SST/macro HPX Implementation

- Almost identical API, so resulting application code is the nearly the same as real HPX
- However, the API is not as filled out as the real thing. Only more commonly used functions/components are implemented
- Default AGAS implemented as distributed hash.
- Do have a model for on-node thread manager for oversubscription, though focus is on the parallel scalability aspects

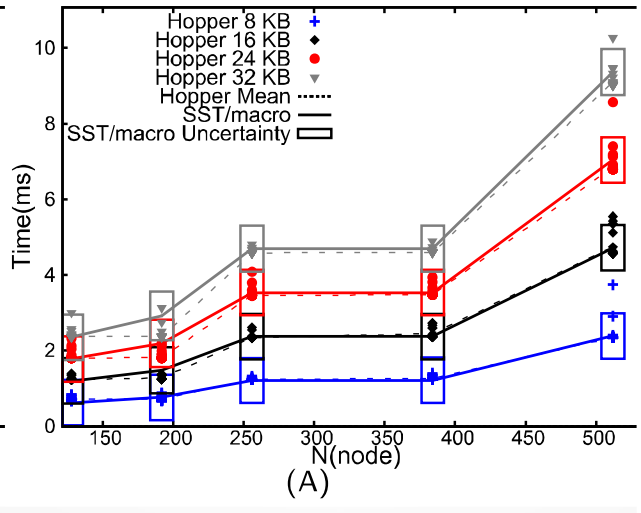
Simulator Validation

- Validation study has been completed against a Cray XE6, for MPI
- Able to capture congestion behavior that results from both hardware and MPI implementation effects
- Demonstrated simulation validation workflow that includes formal UQ methods

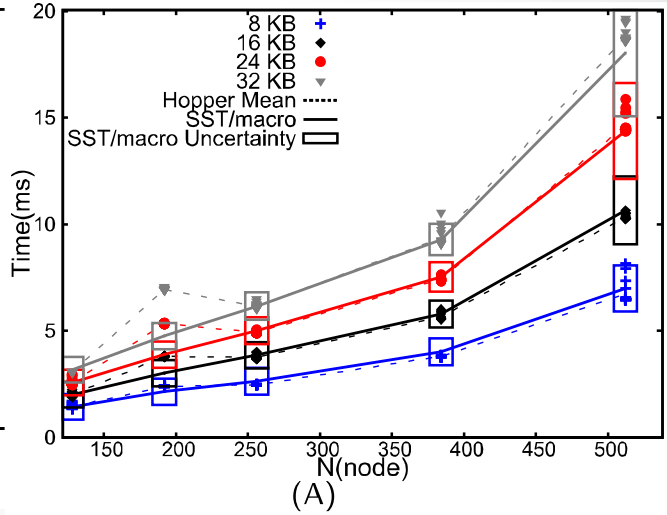
MPI_Scatter



MPI_Gather

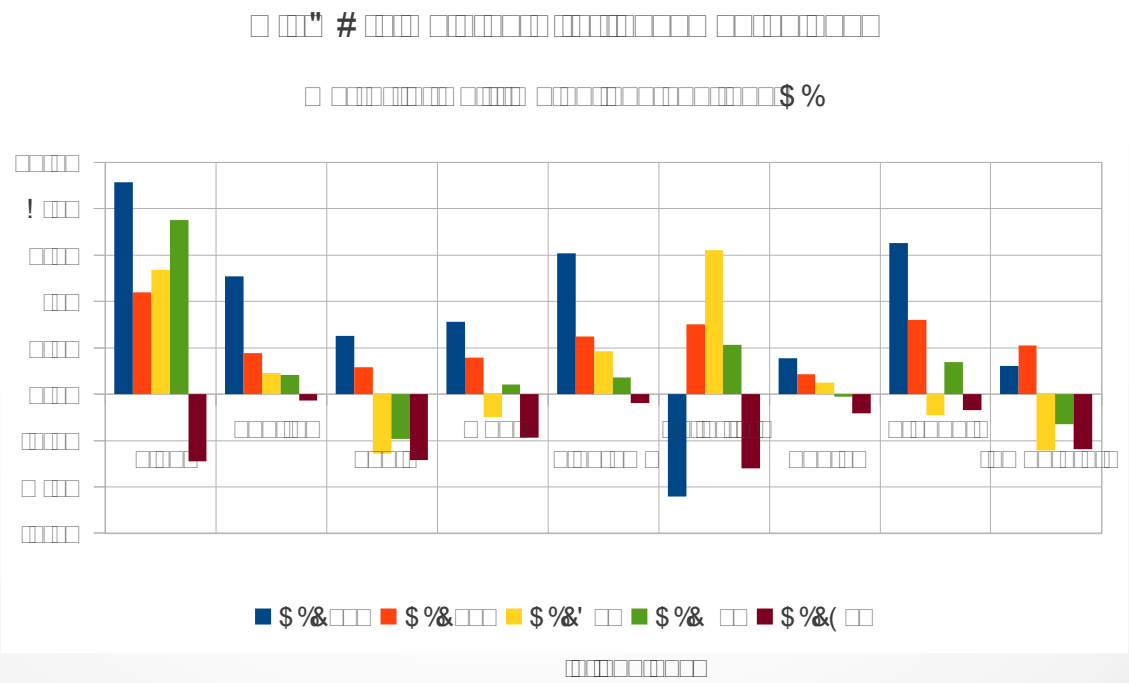


MPI_Allgather



Computation Modeling

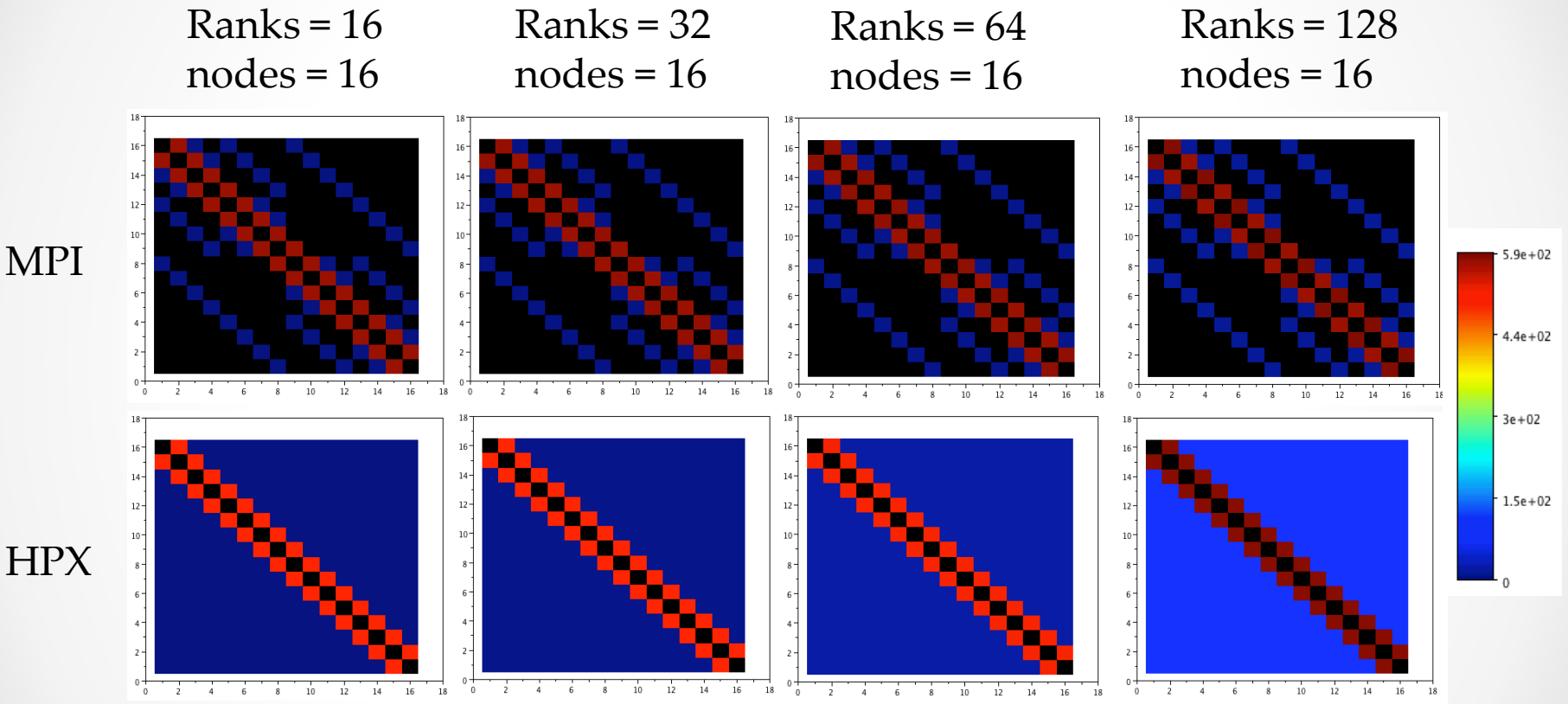
- Online simulation using user-space threads, means simulation time doesn't pass between library calls
- need computation modeling between them
- Naïve linear model based on nested loop bounds
  - parameterized at runtime, but still very simple
- Preliminary results are still very promising:



Results on HPCCG

- Use it because it is extremely simple
  - it's not very predictive of any app, lacks real context around the conjugate gradient solver
  - but it's easy to understand, and easy to work with
- Compare general application characteristics using MPI and HPX
  - So we can know what the application is doing, runtime characteristics
- Explore sensitivity of EM to architectural parameters

Collectives make communication patterns different

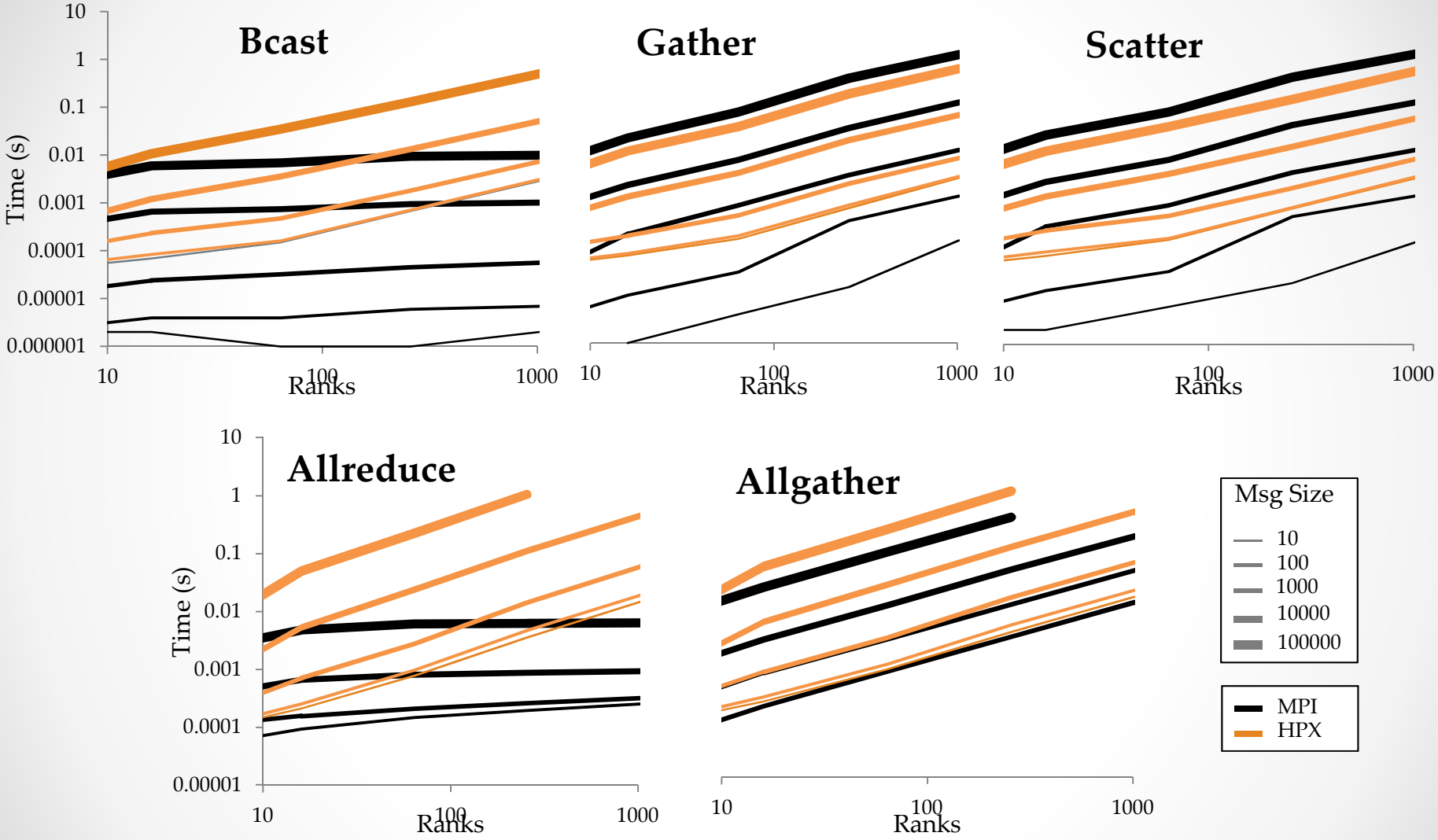


- HPCCG is mostly nearest-neighbor in 1D
- HPX collective implementation is all-to-all to exploit asynchrony
  - produces a lot of traffic

Let's ask a more fundamental question

- MPI collectives are optimized for reducing network traffic
- HPX collectives can be implemented to maximize asynchrony, or they can be implemented to minimize communication
- What is the performance implications of these two as they scale?
  - In other words, if we don't reformulate algorithms or porting an application so that collectives aren't blocking, how will HPX perform?

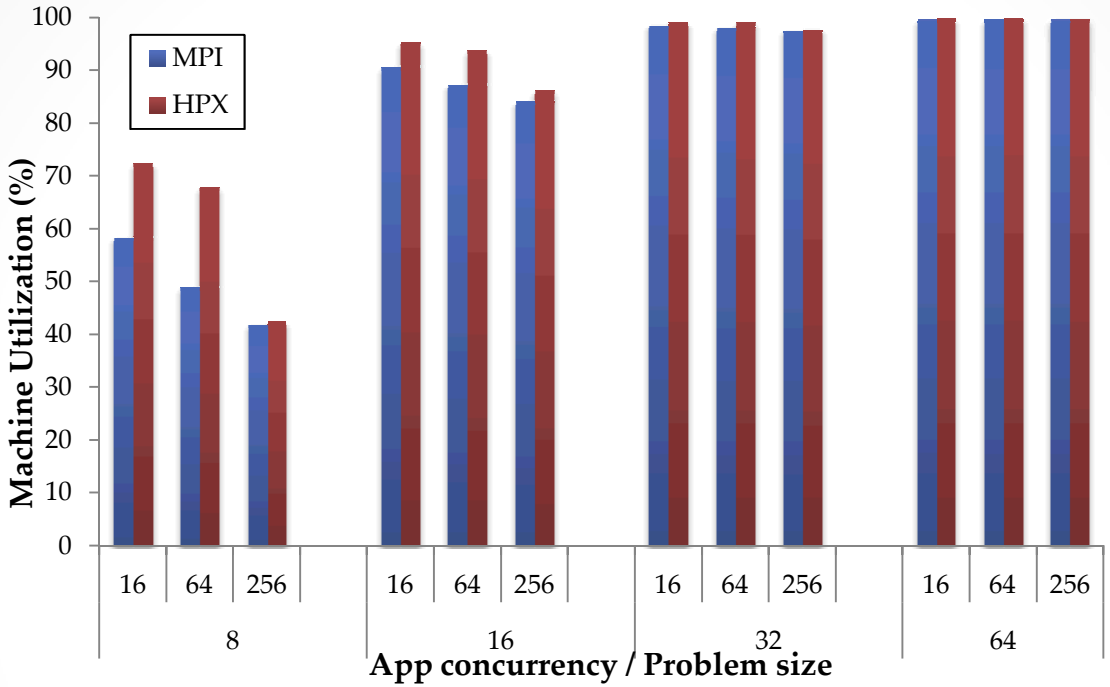
Zero'th order Collective comparisons



- 10 blocking collectives with 10ms of compute time each
- Bcast and Allreduce don't scale well in HPX
- Gather and Scatter do scale well in HPX
- Allgather is ok, some overhead

Sandia National Laboratories

Exploiting asynchrony in communication-bound problem

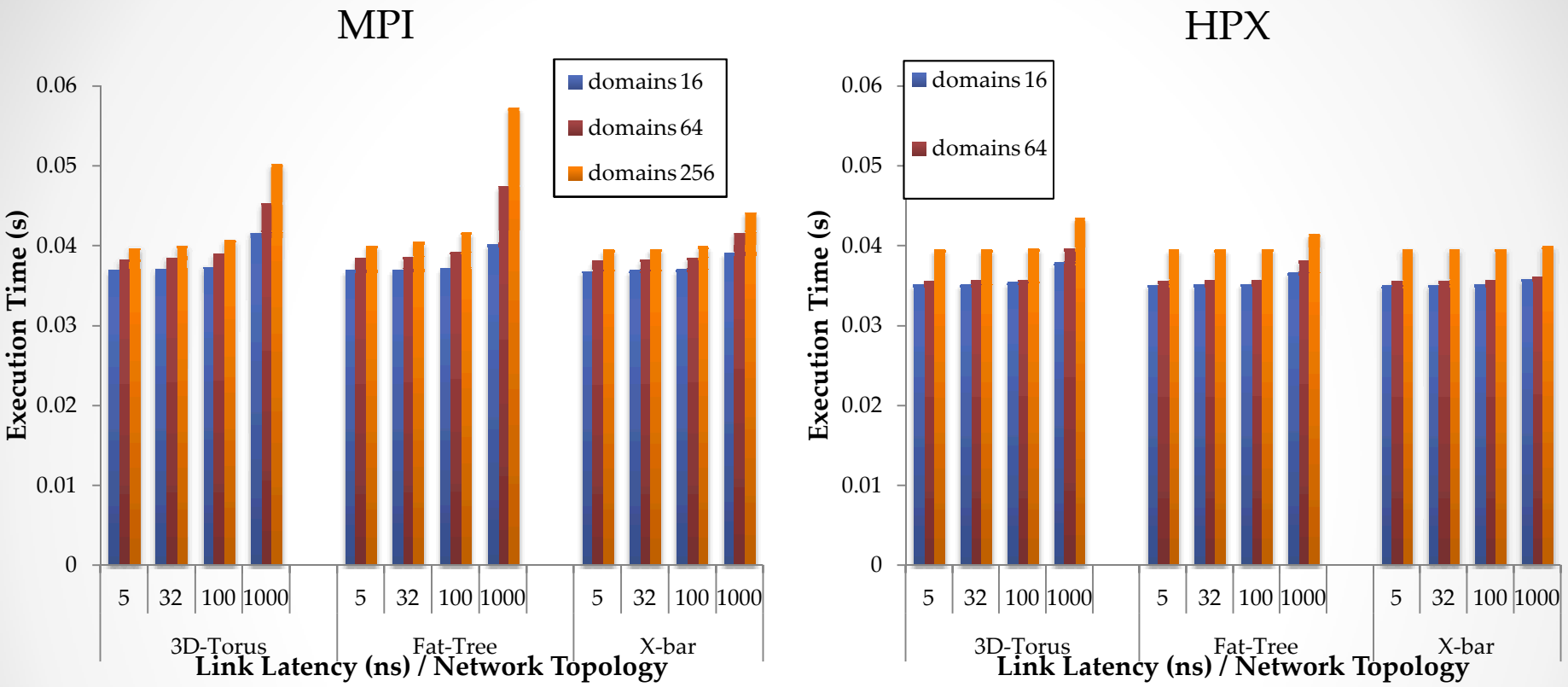


- Measuring machine utilization (avg compute time / avg total time) for different problem sizes and application concurrencies
- HPCCG has no problem weak scaling
  - we knew that, we're just playing with it because it's easy
- HPX does exploit some asynchrony for smaller problem sizes
- Larger problem sizes are compute-bound anyway

Some small experiments with HPCCG

- Compare general application characteristics using MPI, HPX, and PGAS
- Explore sensitivity of EM to architectural parameters
  - Figure out which ones are important for each EM
  - Some key parameters:
    - network latency, bandwidth, topology
    - memory bandwidth

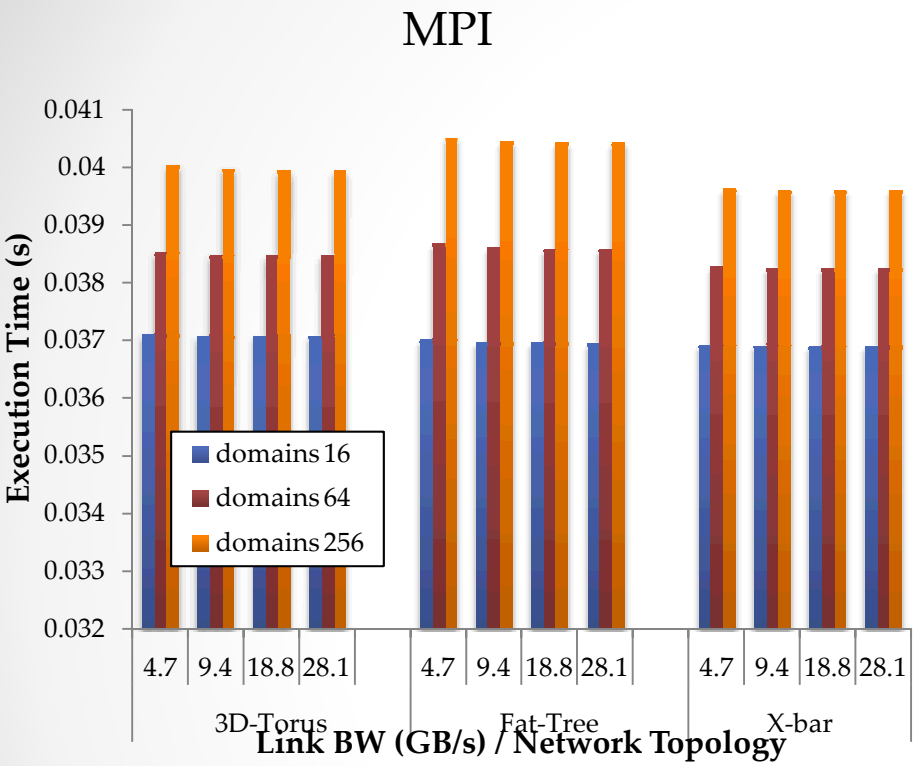
network link latency

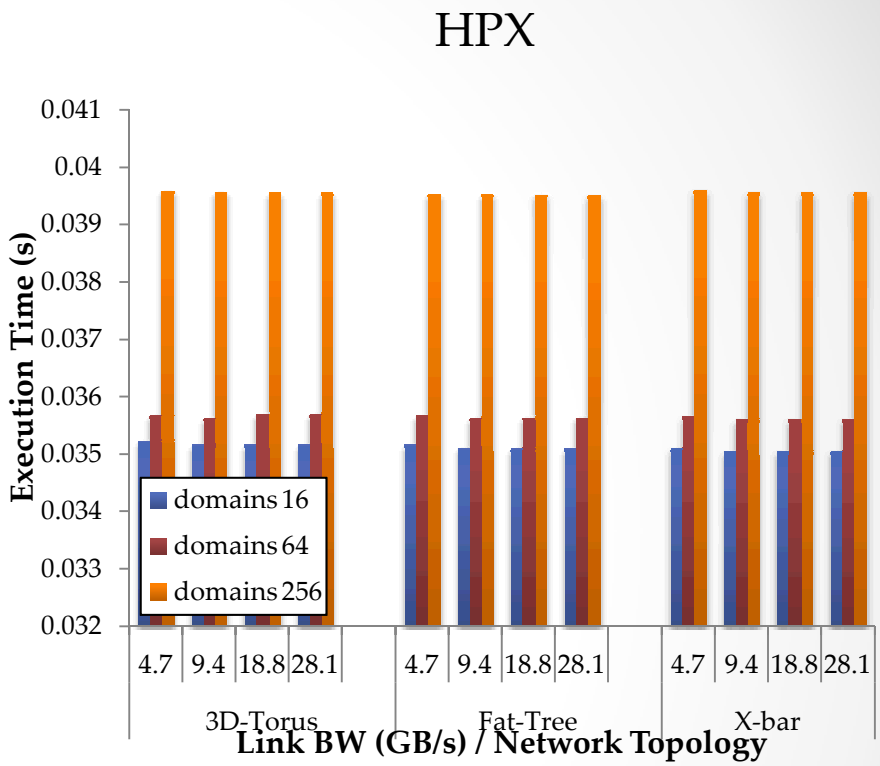


- HPX looks less sensitive to link latency
  - this is expected from asynchronous collectives, what HPX is after
- Also less sensitive to topology
-

Sandia National Laboratories

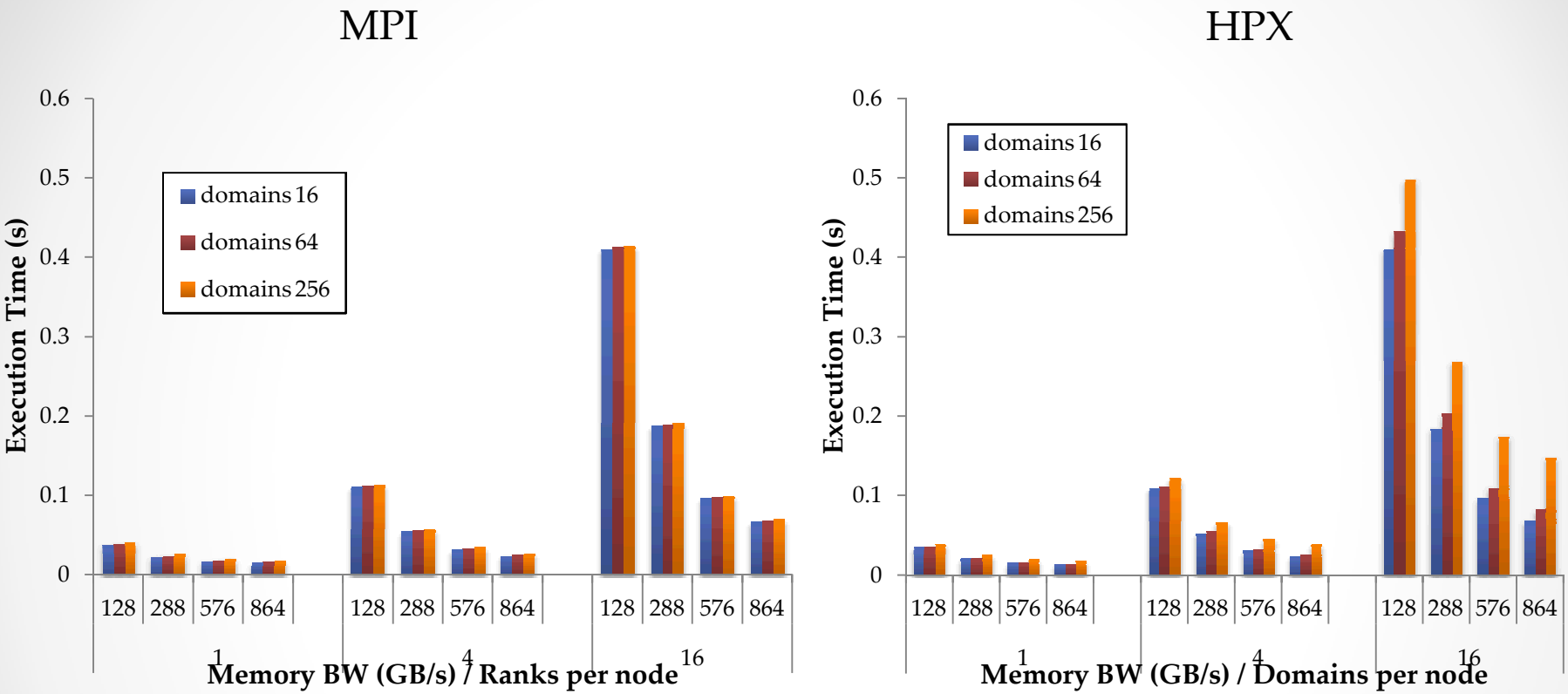
BW





- HPCCG not really sensitive to link bw
  - could be limited by something else

Memory BW



- More sensitive as you put more ranks/domains on a node
- HPX more sensitive than MPI

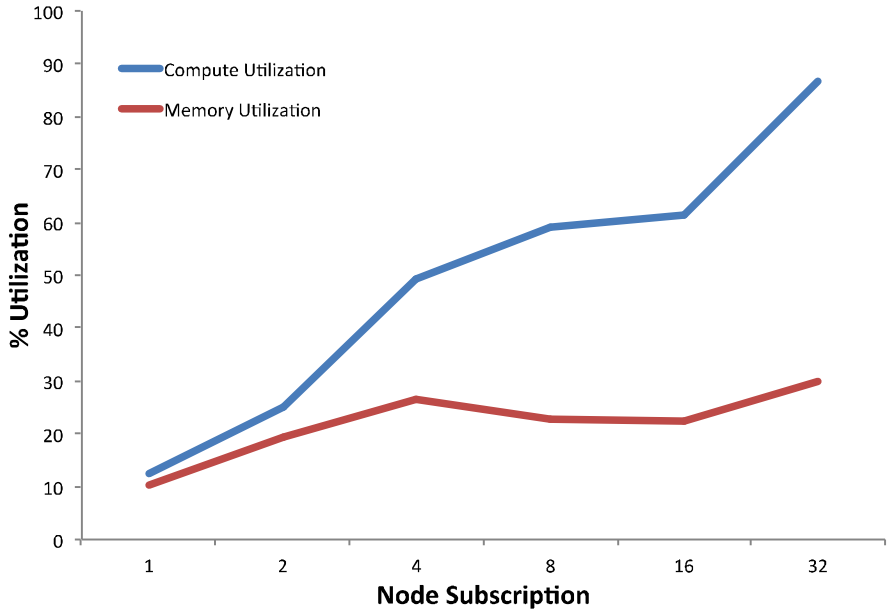
Sandia National Laboratories

Results on GTC

- Original zero'th order port
  - latest version has staged collectives, some functions pipelined as futures

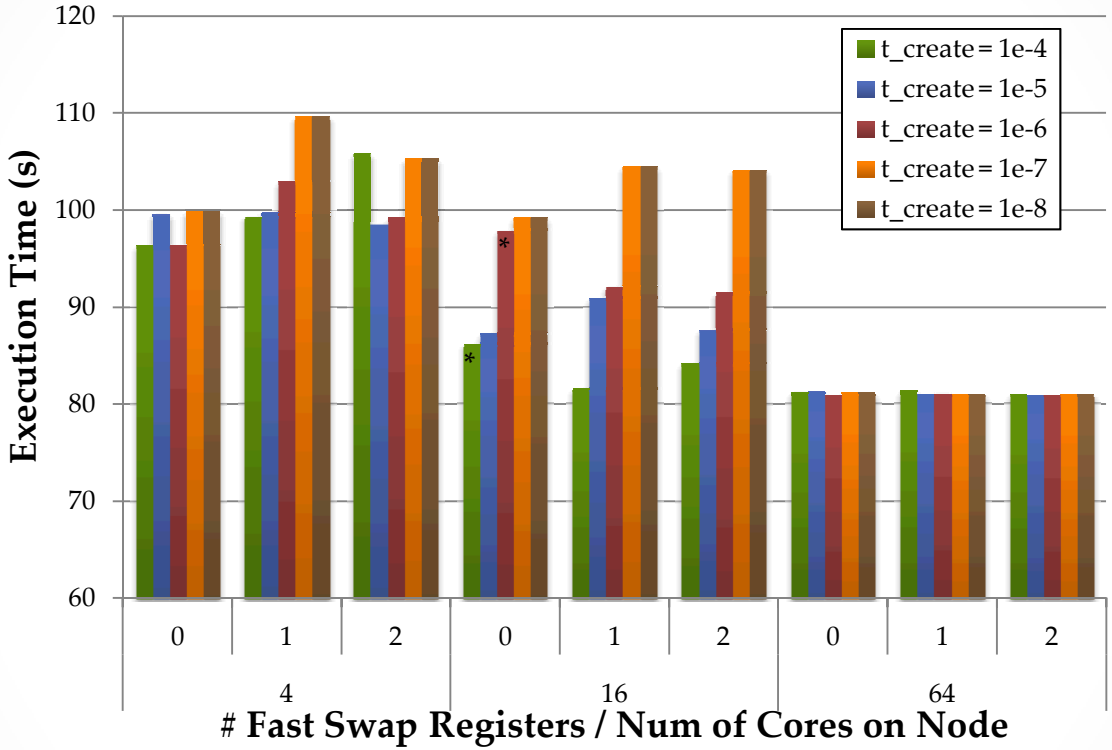
- Try to perform some of the on-node experiments that were meant to be done by lower level simulators
  - thread overheads, etc.

Resource Utilization



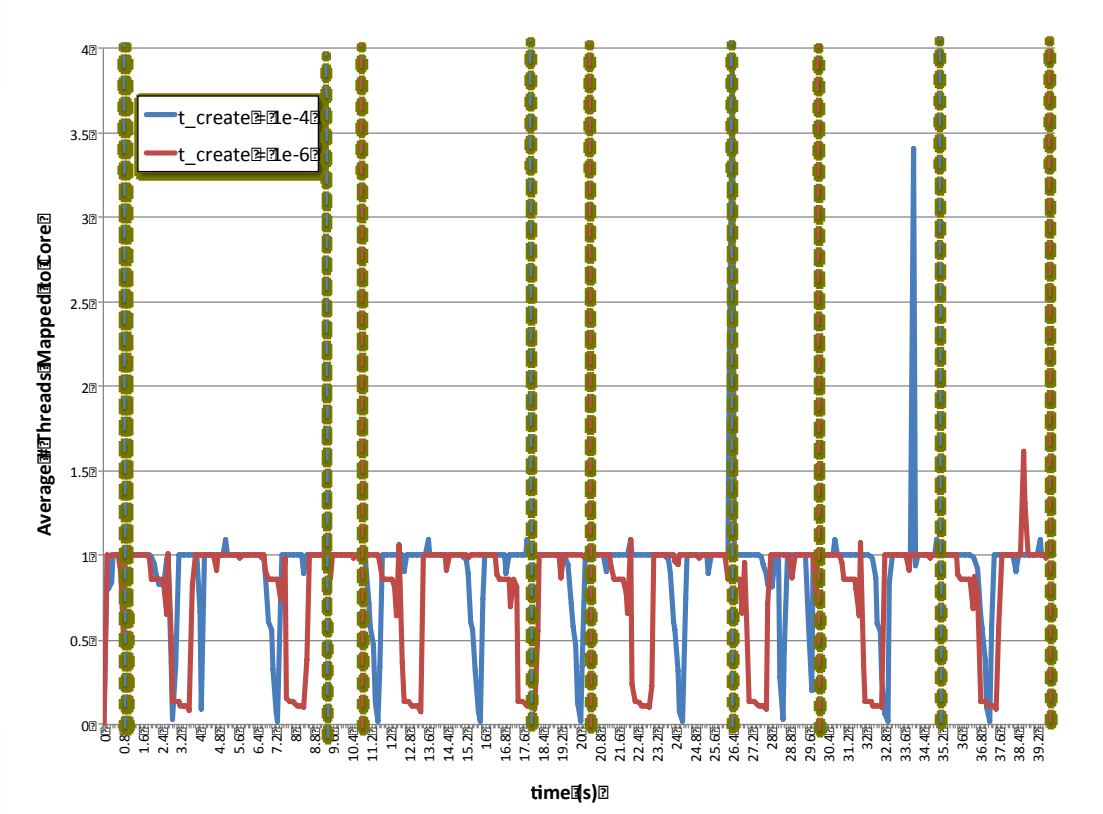
- Increasing node subscription maxes out memory bandwidth

Node/core characteristics



- If anything, highlights the imperfection of our memory contention model

Still interesting execution visualizations



- see how execution gets delayed
- dashed lines = iteration boundaries
-

Technical Summary

- HPX is capable of exploiting global asynchrony to the extent possible
  - but use staging for Allreduces and Bcasts or scalability will kill you
- HPX can be less sensitive to network parameters through latency hiding, which may be good considering the unknown exascale landscape

Programmatic Lessons Learned Thus Far

- Porting applications is non-trivial (but we already knew that).
  - It requires re-formulating the solvers
  - It is not reasonable to assume an CSP app will perform optimally in async task EM w/o a major rewrite
  - The project should have included some miniApps up front to develop methodology and prove out our approach and infrastructure
- Wish we had cycle-accurate simulation results to make conclusions about on-node tradeoffs

Future Work

- validate HPX implementation
- more apps, reformulation
- complete PGAS implementation in simulator (and validate) as a baseline alternative
  - OpenSHMEM is like an in-between point between MPI and HPX in that it has an asynchrony feature (one-sided communication) and a global naming feature (symmetric data). But it's still structured, and porting is pretty easy.
- MPI vs PGAS vs HPX showdown for some apps at larger scales:
  - GTC, Nekbone, HPCCG, CNS

U.S. DEPARTMENT OF ENERGY | Office of Science

Objectives of the Modeling Execution Models Project

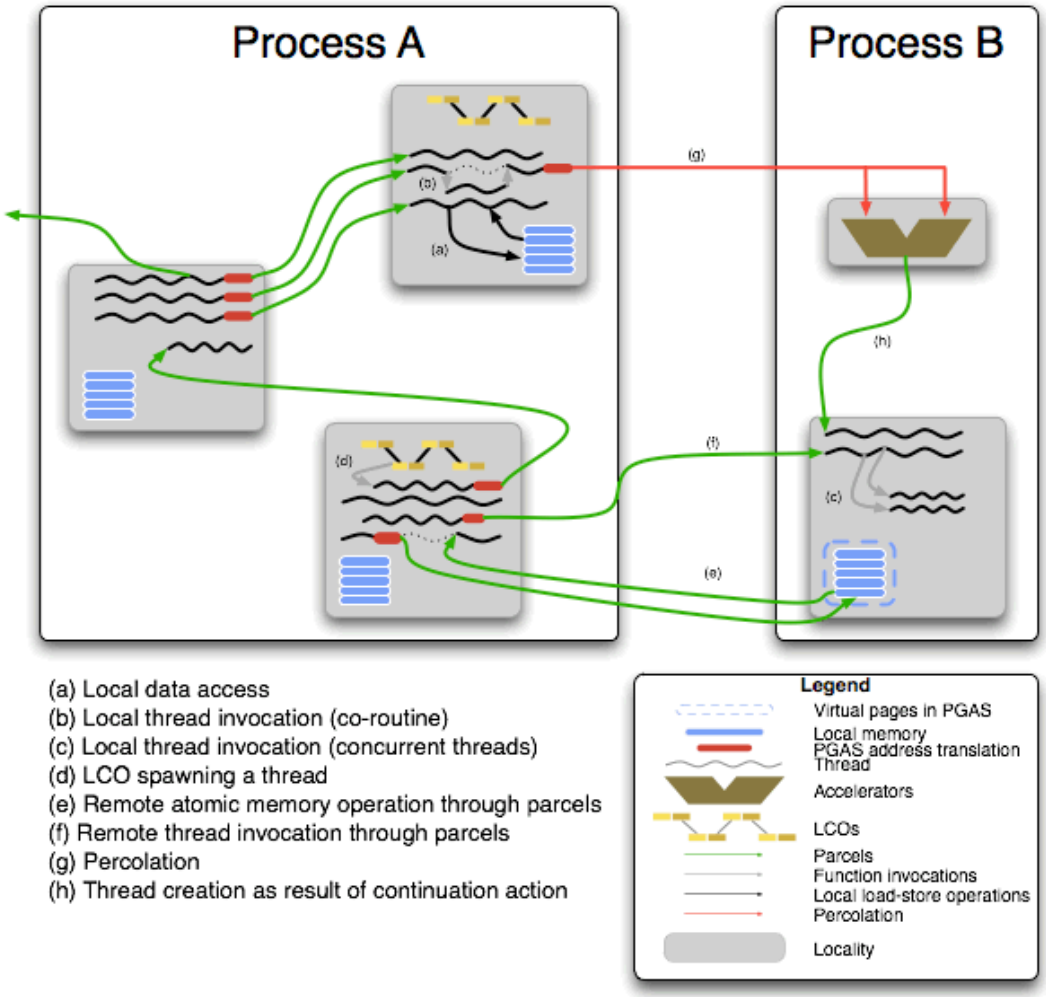
Thomas Sterling, Indiana University

Strategic Objectives

- **Model execution model abstract performance tradeoff space and identify domains of operation opportunities**
  - Use of numeric performance model across parameter ranges
- **Experiments with DOE relevant codes (e.g., proxy-apps) to demonstrate comparative performance advantage**
  - Compare with respect to conventional practices (e.g., MPI)
  - Use codes including GTC, Neckbone, MiniFe, and others
  - Show improvements in efficiency and scalability
- **Quantitative determination of causes of performance differences to identify and understand contributing source factors**
  - Use of SST/micro simulator
  - Measure key overhead costs and achieved utilization
  - Measure scalability achieved through greater exposed parallelism

ParalleX Execution Model

- Lightweight multi-threading
  - Divides work into smaller tasks
  - Increases concurrency
- Message-driven computation
  - Move work to data
  - Keeps work local, stops blocking
- Constraint-based synchronization
  - Declarative criteria for work
  - Event driven
  - Eliminates global barriers
- Data-directed execution
  - Merger of flow control and data structure
- Shared name space
  - Global address space
  - Simplifies random gathers



Basic ParalleX Primitives

- Thread
- Parcel
- LCO
- AGAS
- Procedures/Processes

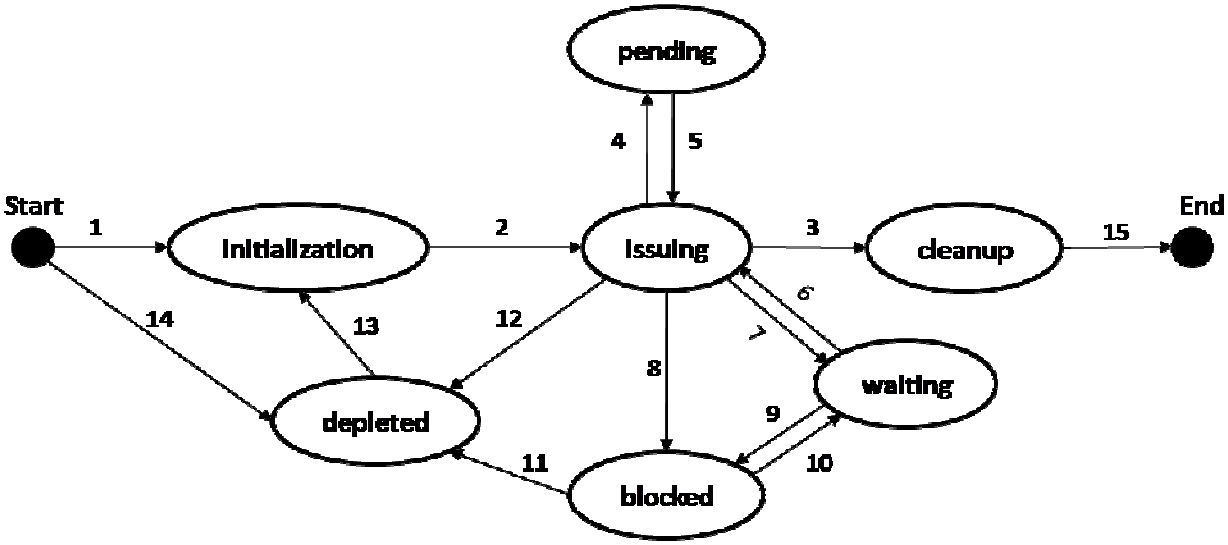
Threads

- Very lightweight unit of work to be performed in a single locality
- Expose parallelism
- Millions can be active at a time
- Fast switching, low overhead
- Provides latency hiding

Complexes, Threadmanager

- Threads are first class objects
  - Have a (globally unique) name
- Lightweight, fine grained
  - Fast context switching between threads
  - Threads yield to other threads while waiting for any resource to get available
    - Latency hiding (memory access, remote access, and service requests)
- Needed:
  - Hardware support for fast context switching
  - Integration with OS scheduler
  - Integration with OS resource management
  - Compiler support

ParalleX Thread States



1. thread creation request
2. thread state initialization complete
3. Request to terminate thread
4. No resource(s) available
5. Resource(s) available
6. No ready operations
7. Operations ready for execution
8. in-flight blocking parcel
9. in-flight parcel arrived
10. No in-flight operations and no pending operations
11. (Policy Decision)
12. External event or defer operations
13. External event
14. Depleted thread creation request
15. Executed stop operation

Parcels

- PARallel Control Element
- Variant of active messages
- Provides a continuation list
- Agent of all remote actions
- Provides latency hiding
- Not a first class object

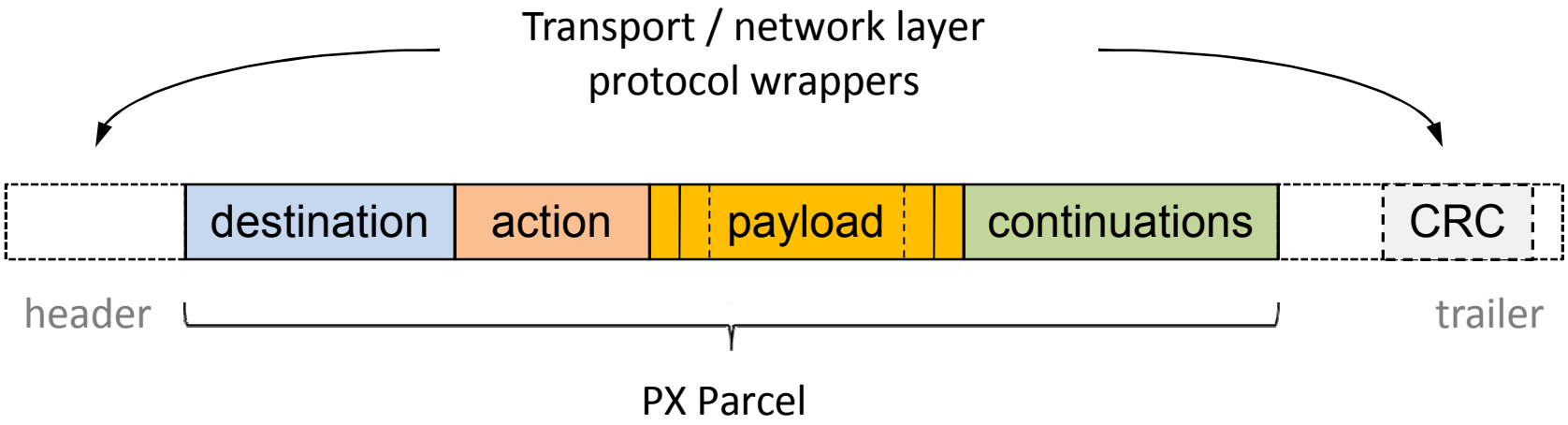
Motivation for Message-Driven Computation

- To achieve high scalability, efficiency, programmability
- To enable new models of computation
  - e.g., ParalleX
- To facilitate conventional models of computation
  - e.g., MPI
- Hide latency
  - Support overlap of communication with computation
  - Move work to data, not always data to work
- Work-queue model of computing
  - Segregate physical resource from abstract task
  - Circumvent blocking of resource utilization
- Support asynchrony of operation
- Maintain symmetry of semantics between synchronous and asynchronous operation

Parcels and the Parcel-sets

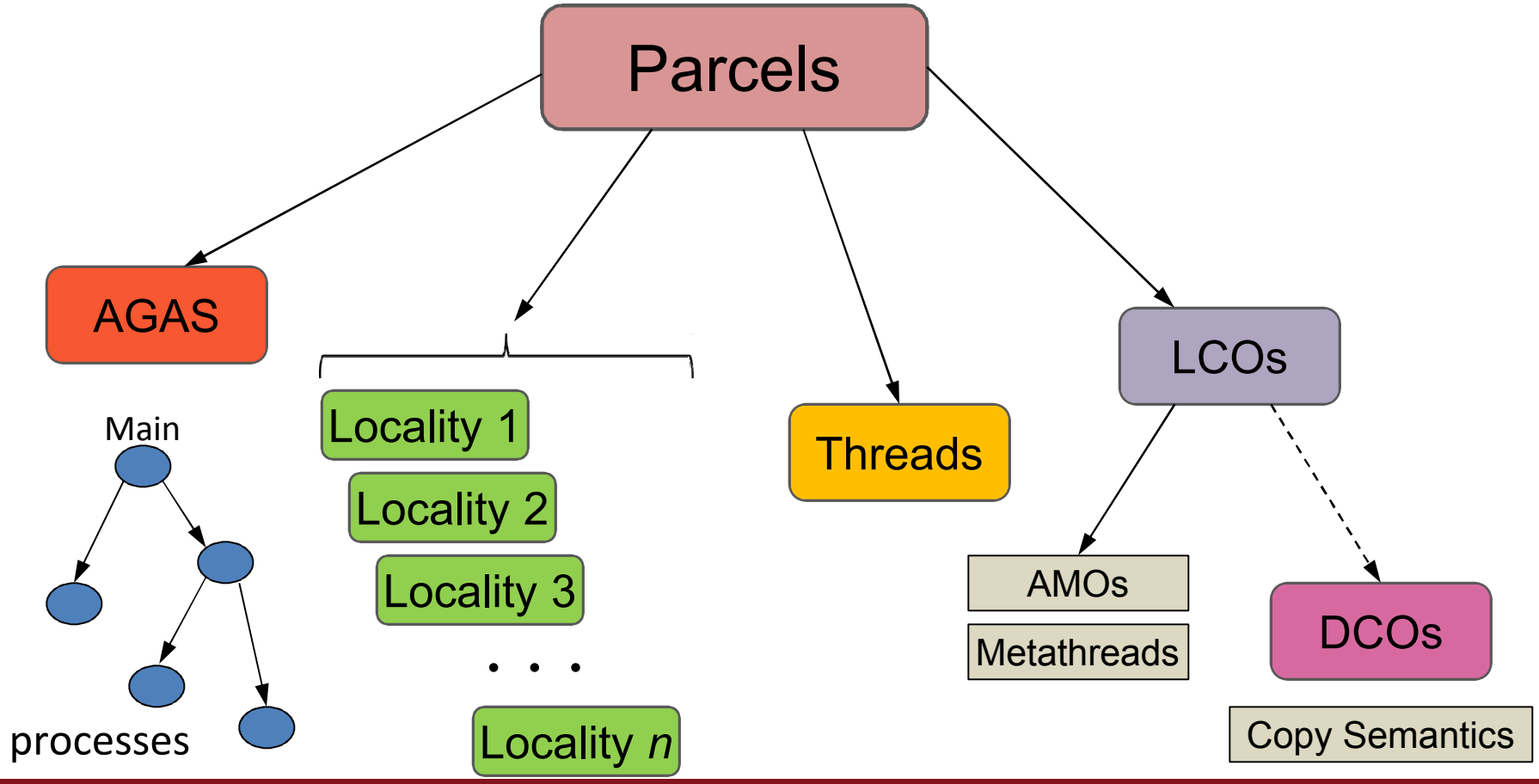
- Parcels are active messages used to
  - Move work to the (remote) data
  - Gather data from remote locations
- Parcel-set is the set of all parcels in the system
  - Representation in each locality
  - Used to put (send) and get (receive) parcels
  - Dynamic routing
- Needed:
  - Hardware support (NICs?)
    - Certain parcels can be handled directly
      - Simple compound atomic operations
  - Operating system
    - Provide network interface to be used

Parcel Structure

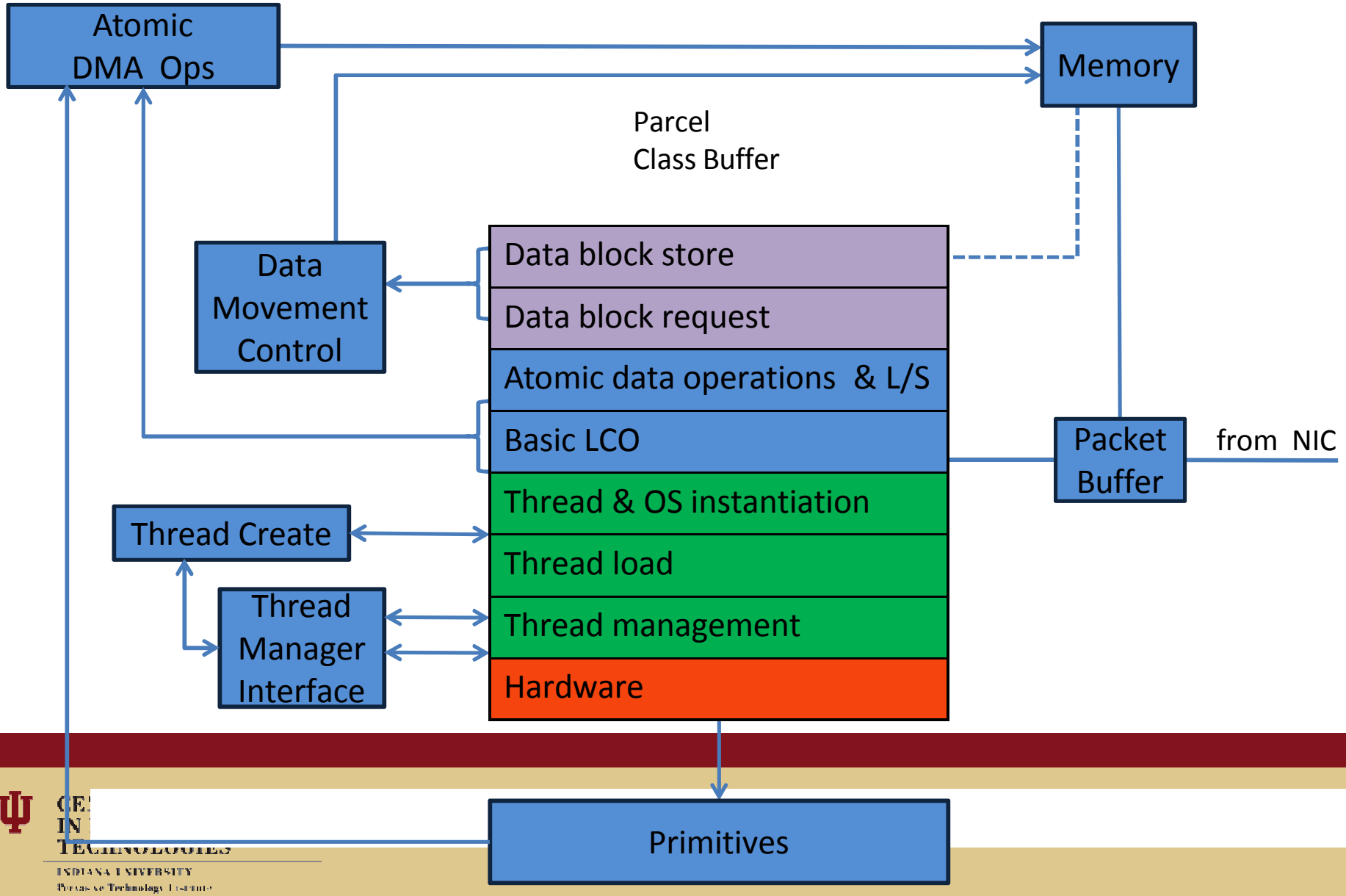


Parcels may utilize underlying communication protocol fields to minimize the message footprint (e.g. destination address, checksum)

Parcel Interaction with the System



Parcel Handler Implementation



LCOs

- A number of forms of synchronization are incorporated into the semantics
- Support message-driven remote thread instantiation
- In-memory synchronization
  - Control state is in the name space of the machine
  - Producer-consumer in memory
  - Local mutual exclusion protection
  - Synchronization mechanisms as well as state are presumed to be intrinsic to memory
- Basic synchronization objects:
  - Mutexes
  - Semaphores
  - Events
  - Full-Empty bits
  - Data flow
  - Futures
  - …
- User-defined (custom) LCOs

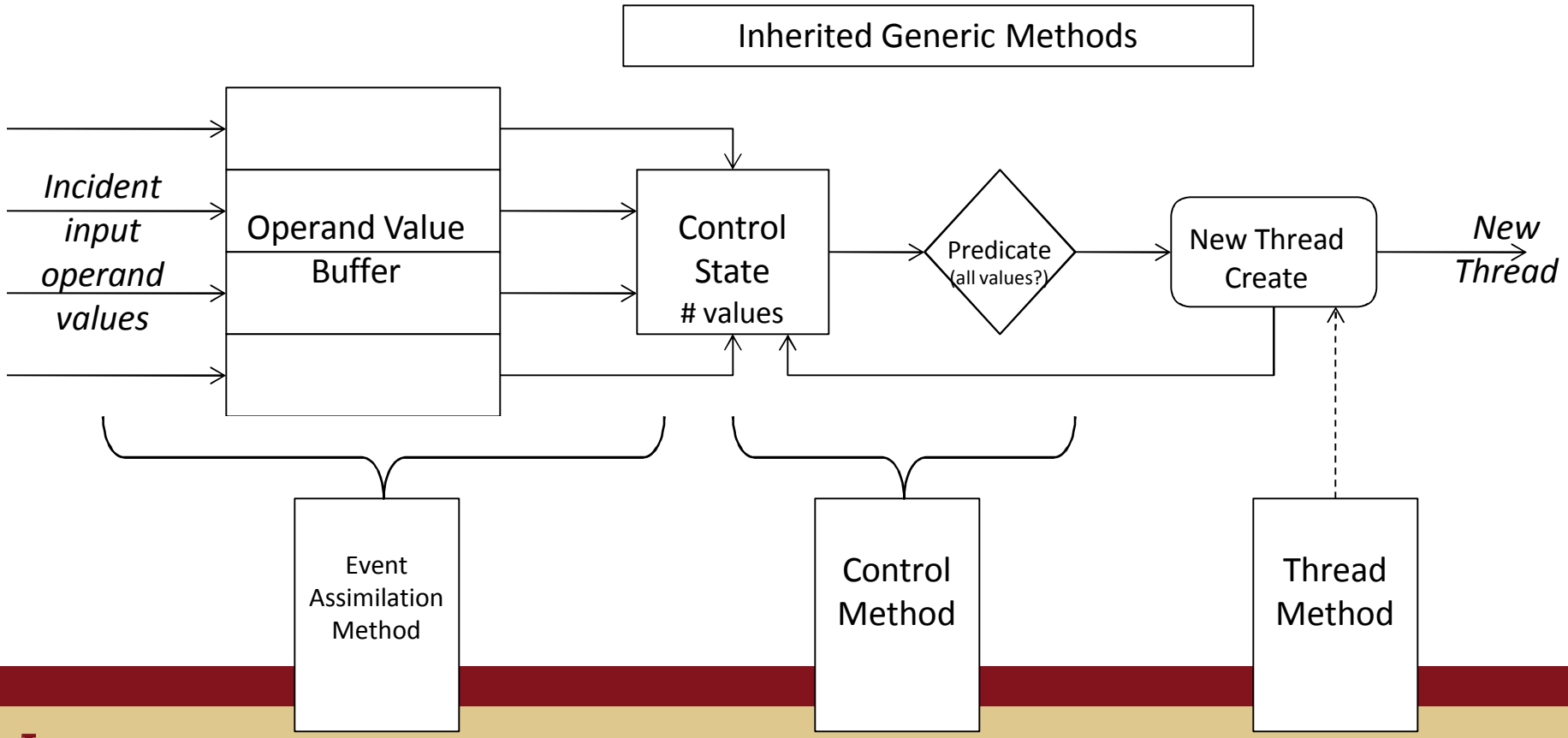
Goals of the ParalleX LCO

- Exploit parallelism in diversity of forms and granularity
  - For extreme scalability
  - e.g., exploit meta-data defined parallelism
- Latency hiding at system-wide distances
  - Avoid conventional round trip control patterns
  - Support latency mitigating architectures
- Provide a framework for efficient fine grain synchronization
  - Eliminate use of global barrier synchronization where possible
  - Mitigate effects of variable thread lengths within fork-join structures
- Enable optimized runtime adaptive resource management and task scheduling for dynamic load balancing
- Migration of *continuations* across system and computation
- Support eager-lazy evaluation methods
- Support distributed control operations
- Semantics of failure response for graceful degradation
  - Used to establish points for "micro-checkpointing" and validation for error detection, propagation isolation, and recovery

Constraint-based Synchronization

- Supports Dynamic-Adaptive Task Scheduling
- Declarative Semantics for Continuation of Execution
  - Defines conditions for work to be performed
  - Not imperative code by user
- Establishes Criteria for Task Instantiation
- Supports DAG flow control representation
- Examples:
  - Dataflow
  - Futures

Dataflow LCO



Localities (Synchronous Domains)

- A “locality” is a contiguous physical domain
- Guarantees compound atomic operations on local state
- Manages intra-locality latencies
- Exposes diverse temporal locality attributes
- Divides the world into synchronous and asynchronous
- System comprises a set of mutually exclusive, collectively exhaustive localities
- A first class object
- An attribute of other objects
- Heterogeneous
- Specific inalienable properties

Active Global Address Space (AGAS)

- Distributed
- Assumes no coherence between localities
- Moves virtual named elements in physical space
- Examples
  - User variables
  - Synchronization variables and objects
  - Parcel sets (but not parcels!)
  - Computational Complexes (Threads) as first-class objects
  - Processes
    - First class object
    - Specifies a broad task
    - Defines a distributed environment
      - Spans multiple localities
      - Need not be contiguous

ParalleX Processes

- Main abstraction encapsulating parallel execution
  - First class objects
  - Contexts for data, threads, and child processes
  - Supports lazy arguments (think subroutines/functions)
  - Functions across multiple nodes
  - Spans namespace for (part of) application
  - Manages access policies (data members might be public/protected/private)
- Capabilties-based Addressing
  - Protection from external security attacks
  - Protection from internal security attacks between UHPC modules
  - Guard access to state resources within ParalleX processes
  - Protocol for transferring access rights between ParalleX processes

ParalleX Process